Alexander Garcia May 30, 2017

Assignment-1

1. Let $f(x) = e^x \cos 2x$.

   (a) The definition of the Taylor polynomial is as follows:

$$T_n(x) = \sum_{k=0}^{n} \frac{(x - x_0)^k}{k!} f^{(k)}(x_0)$$

   In this case, $n = 4$, and $x_0 = 0$.
   We start by finding $f^{(1)}(x), f^{(2)}(x), f^{(3)}(x)$, and $f^{(4)}(x)$

$$f^{(1)}(x) = -2e^x sin(2x) + e^x cos(2x) = e^x(cos(2x) - 2sin(2x))$$
$$f^{(2)}(x) = e^x(-2sin(2x) - 4cos(2x)) + e^x(cos(2x) - 2sin(2x)) = -e^x(4sin(2x) + 3cos(2x))$$
$$f^{(3)}(x) = -e^x(8cos(2x) - 6sin(2x)) - e^x(4sin(2x) + 3cos(2x)) = e^x(2sin(2x) - 11cos(2x))$$
$$f^{(4)}(x) = e^x(4cos(2x) + 2sin(22sin(2x)) + e^x(2sin(2x) - 11cos(2x)) = e^x(24sin(2x) - 7cos(2x))$$

   Evaluating these functions at the center point, we get:

$$f(0) = e^0 cos(0) = 1$$
$$f^{(1)}(0) = e^0(cos(0) - 2sin(0)) = 1$$
$$f^{(2)}(0) = -e^0(4sin(0) + 3cos(0)) = -3$$
$$f^{(3)}(0) = e^0(2sin(0) - 11cos(0)) = -11$$
$$f^{(4)}(0) = -e^0(24sin(0) - 7cos(0) = -7$$

   We now have all the pieces we need to determine the polynomial expansion.

$$P_n(x) = \frac{x^0}{0!} f(0) + \frac{x^1}{1!} f^{(1)}(0) + \frac{x^2}{2!} f^{(2)}(0) + \frac{x^3}{3!} f^{(3)}(0) + \frac{x^4}{4!} f^{(4)}(0)$$

$$= 1 + x - \frac{3}{2}x^2 - \frac{11}{6}x^3 - \frac{7}{24}x^4$$

   (b) By definition, the derivitive form of the remainder is

$$R_n(x) = \frac{(x - x_0)^{n+1}}{(n+1)!} f^{(n+1)}(c)$$

   where c is close to x_0.
   In this case, we have $n = 4, x_0 = 0$ from the previous question.

$R_n(x) = \frac{x^5}{5!} f^{(5)}(c)$
$f^{(5)}(c) = e^x(48cos(2x) + 14sin(2x)) + e^x(24sin(2x) - 7cos(2x))$
$= e^x(41cos(2x) + 38sin(2x))\,]_{x=c}$

Therefore,

$$R_4(x) = \frac{x^5}{120} e^c(41cos(2c) + 38sin(2c))$$

(c) We know the remainder, $R_4(x)$ from part (c). Given a closed interval, we can find the maximum error by finding the maximum remainder in this interval. To do this, we must find its derivative, whose zeroes will show all maxima and minima.
$f^{(6)}(x) = e^x(-82sin(2x) + 76cos(2x)) + e^x(41cos(2x) + 38sin(2x)) = e^x(117cos(2x) - 44sin(2x))$

Now find zeroes.
$e^x(117cos(2x) - 44sin(2x)) = 0$
$117cos(2x) - 44sin(2x) = 0$
$\frac{117}{44} = \frac{sin(2x)}{cos(2x)}$
$x = \frac{1}{2} tan^{-1}(\frac{117}{44}) \approx 0.606$

Our candidate points are then $\{-\frac{\pi}{4}, 0.606, \frac{\pi}{4}\}$

$f^{(5)}(\frac{\pi}{4}) = 38 * e^{\frac{\pi}{4}} \approx 83.3$
$f^{(5)}(-\frac{\pi}{4}) = -38 * e^{\frac{\pi}{4}}$
$f^{(5)}(0.606) = e^{0.606}(41cos(2 * 0.606) + 38sin(2 * 0.606)) \approx 91.6$

As $f^{(5)}(0.606)$ is clearly the max, this is what we use for determining the error bound.

$$max(R_4(x)) = \frac{(\frac{\pi}{4})^5}{5!} * 91.6$$

$$\approx 0.228$$

(d) Figure 1 shows the actual function compared to its 4^{th} degree Taylor expansion over the range $x \in [\frac{-\pi}{4}, \frac{\pi}{4}]$. Figure 2 shows the maximum error between the two, which agrees with the upper bound calculated in part (c).

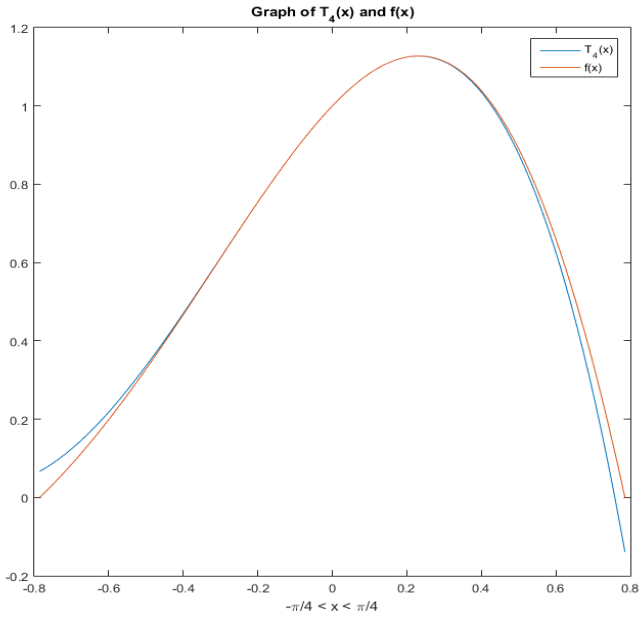

Figure 1: Taylor expansion and initial function

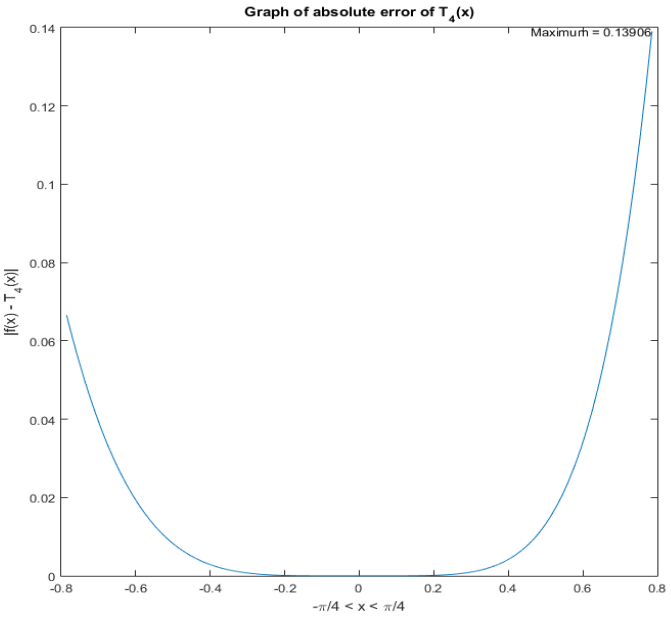

Figure 2: Difference between the Taylor expansion and the initial function

2. $b = 10.\overline{110}_2 \ = 2^1 + (0.\overline{110})_2$

   $z = 0.\overline{110}_2$

   $z = 0.110\overline{110})2$

   $2^3 * z = 110.\overline{110}_2 = 2^2 + 2^1 + z$

   $8z = 6 + z$

   $z = \frac{6}{7}$

   $b_{10} = 2 + z_{10} = 2 + \frac{6}{7} = \frac{18}{7} = 2.\overline{857142}$

3. 6.7_{10}

   $6_{10} = 110_2$

   $\frac{14}{10} = d_1 + \frac{d_2}{2} + \cdots \quad d_1 = 1$
   $\frac{4}{5} = d_2 + \frac{d_3}{2} + \cdots \quad d_2 = 0$
   $\frac{8}{5} = d_3 + \frac{d_4}{2} + \cdots \quad d_3 = 1$
   $\frac{6}{5} = d_4 + \frac{d_5}{2} + \cdots \quad d_4 = 1$
   $\frac{2}{5} = d_5 + \frac{d_6}{2} + \cdots \quad d_5 = 0$
   $\frac{4}{5} = d_6 + \frac{d_7}{2} + \cdots \quad d_6 = 0$

   The value generating d_6 is the same as the one generating d_2, and thus we are in a loop generating the repeating pattern 0110.

   $6.7_{10} = 110.1\overline{0110}_2 = 1.101\overline{0110} * 2^2$

   The IEEE representation of this number designates one signed bit (in this case it is 0, as 6.7¿0), 8 for the exponent (in this case 2), and 23 for the mantissa (in this case $101\overline{0110}$. The 23^{rd} digit of the mantissa is 0, so the number is truncated when stored.

   The exponent that is stored is the real exponent + a bias, which in this case is 127.

   $fl(6.7)$ is stored in the machine as

0	10000001	10101100110011001100110
sign	exponent + bias	mantissa

   Converting the stored number back to decimal gives us $2^{129-127} * q$

   $q = 1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-6} + 2^{-9} + 2^{-10} + 2^{-13} + 2^{-14} + 2^{-17} + 2^{-18} + 2^{-21} + 2^{-22} \approx 1.675$

   $fl(6.7) = 2^2 * q \approx (6.7 + 1.90735 * 10^{-7})$

   The relative error $d = \frac{6.7 - fl(6.7)}{6.7} = 2.84679 * 10^{-8}$

   ϵ_{mach} for a 32 bit float is $\frac{1}{2^{23}} \approx 1.19 * 10^{-7}$

   $\frac{1}{2} * \epsilon_{mach} \approx 5.96 * 10^{-8}$

   Crucially,

$$\frac{1}{2} * \epsilon_{mach} \leq d$$

4. Holmes 1.5

   (a) According to the IEEE double precision standard, 16_{10} is stored as $1 * 2^5 =$

0	10000000100	$0 \cdots 0$
sign	exponent + bias of 1023	52 bit mantissa

If we let d_n represent the n^{th} digit in the mantissa, then the last digit stored in the machine is $d_{52} = 0$. Beucase the first 4 digits of the mantissa are used to make up the whole number 16, there are 48 binary digits available to represent the decimal. If the 49^{th} digit is a 1, and all $[d_{50}, d_\infty)$ are 0, then the round-to-nearest rule states that we chop at the 48^{th} digit, effectively rounding down. This would make the upper bound

$$R = 16 + 2^{-49} = 16.0000000000000017763568394002504646778106689453125$$

The lower bound of x that is recognized as 16 is $15.xxx$, or the lowest value that rounds up to 16. The whole number 15 is represented in binary as

0	10000000010	$1110\cdots0$
sign	exponent + bias of 1023	52 bit mantissa

Unlike the upper bound, it only takes 3 bits of the mantissa to represent the whole number part of the number. There are now 49 binary digits available to represent the decimal. Again using the round to nearest rule, if the 50^{th} digit is a 1, and there is at least one nonzero digit in $[d_{51}, d_\infty)$, then the number is rounded up. This would make the lower bound

$$L > 16 - 2^{-50} \rightarrow L > 15.9999999999999991118215802998747676610946655273437 5$$

$$L < x \leq R \rightarrow x = 16$$

(b) The same process can be used to find the upper and lower bounds of 50.
$50 = 110010 = 1.10010 * 2^5$

0	10000000100	$100100\cdots0$
sign	exponent + bias of 1023	52 bit mantissa

In this case, the whole number part of the number takes 5 bits of the mantissa, leaving 47 digits for the decimal. Using the same logic as part(a) for the upper bound, we need d_{48} to be 1, and $[d_{49}, d_\infty)$ to be 0. This would make the upper bound

$$R = 50 + 2^{-48} = 50.000000000000003552713678800500929355621337890625$$

Here, the lower bound would be $49.xxx$, where all decimal places in the binary form are 1,again through the same logic as part(a).

$49 = 110001 = 1.10001 * 2^5$

0	10000000100	$100010\cdots0$
sign	exponent + bias of 1023	52 bit mantissa

Again, 5 bits of the mantissa are reserved for the whole number 49. The smallest number that will be rounded up is one with $d_{48} = 1$ and "$d_\infty = 1$", or just any number greater than $50 - 2^{-48}$, since there is no last digit. The lower bound is then

$$L > 50 - 2^{-48} \rightarrow L > 49.999999999999996447286321199499070644378662109375$$

$$L < x \leq R \rightarrow x = 50$$

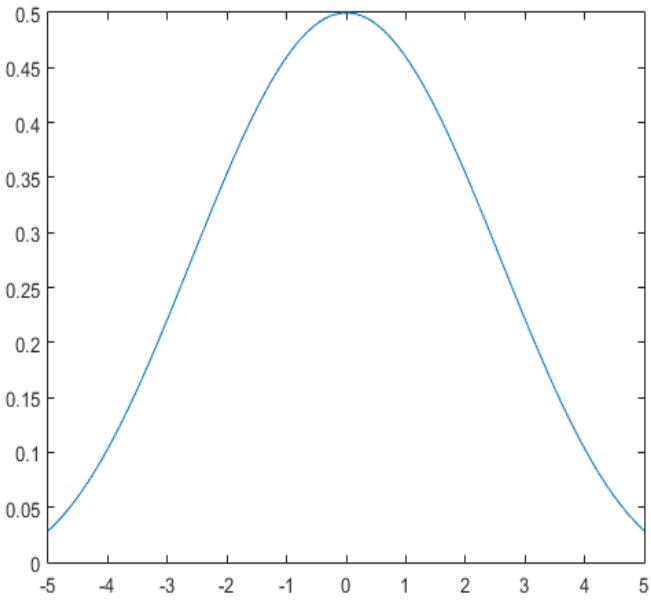

Figure 3: Graph of $\frac{1-cos(x)}{x^2}$

5. It is known that $cos(0) = 1$, so as $x \to 0^+$, the numerator of this function becomes extremely small. When the machine subtracts these two quantities, there is a point at which catastrophic collision begins, which introduces significant error. When the number of ones in the binary form of $cos(x)$ is greater than 52 (the maximum number of digits in an IEEE double precision number), the result will dip below the underflow limit and is seen by the machine as zero.

$$cos(10^{-8}) \approx 0.9999999999999999_{10} \approx 0.111\cdots101_2$$

At $x = 10^{-8}$, there are 54 leading ones in the binary form of $cos(x)$, so the machine computes the numerator as zero. The denominator, however, can still be accurately computed as x^2, so the result is simply 0. At slightly larger values of x ($x = 10^{-7}$), there are less than 52 ones, so the machine can still do meaningful subtraction, albeit with minimal significance in the result.

6. Assume single-precision IEEE arithmetic. Assume that the round-to-nearest rule is used with one modification: if there is a tie then the smaller value is picked (this rule for ties is used to make the problem easier).

   (a) In order for $x > 1$ to hold, the machine must see x as at least $1 + \epsilon_{mach}$. For a single precision machine, $\epsilon_{mach} = \frac{1}{2^{23}} \approx 1.19 * 10^{-7}$. We must now find the lowest possible value of x needed for the machine to make the assignment. According to the round to nearest rule, a float will be rounded up if the binary representation of x has a 1 in the 24^{th} decimal place, and digits $[25, \infty)$ hold at least one nonzero digit. The smallest binary value of x that satisfies these requirements is

   $$x_1 = 1.000\ 000\ 000\ 000\ 000\ 000\ 000\ 001\ 0\cdots1_2$$

   As this number contains an infinite number of zeros after between the final two ones, we can only say that x must be greater than x_1. Therefore,

   $$x > 1 + 2^{-24} \to x > 1.00000005960464477539062 5$$

In order for $x < 2$ to hold, the machine must see x as at most $2-\epsilon_{mach}$. The maximum possible value of x that is seen as $2-\epsilon_{mach}$ is the upper bound on the inequality. According to the round to nearest rule, a float will be rounded down if the final digit in the binary representation of x has a 1 in the 24^{th} decimal place, and digits $[25, \infty)$ are all zero. This is the maximum possible value that can be rounded down, which can be written as

$$x_2 = 2 - 2^{-24} = 1.99999994039535522460937 5$$

The values for which $1 < x < 2$ will hold on a single precision machine are

$$(1.00000005960464477539062 5, 1.99999994039535522460937 5]$$

(b) In order for $x = 4$ to hold, x must be within the machine's "margin of error". The lower bound of this number $(3.999\cdots)$ will take up 1 digit in the mantissa for the whole number, while the upper bound $(4.000\cdots)$ will take up 2 digits. The lower bound, x_1 will then be rounded up if it's 23^{rd} decimal digit is a 1, and there is one other nonzero digit after it.
$x_1 > (4 - 2^{-23}) \rightarrow x_1 > 3.99999988079071044921875$.

The maximum possible value of x that the machine will store as 4 would be $4 + 2^{-22}$, as a 1 in the 22^{nd} decimal place of the binary representation, followed by all zeros is rounded down.
$x_2 = 4 + 2^{-22} \rightarrow x_2 = 4.000000238418579101562 5$.

$$x_1 < x \leq x_2 \rightarrow x = 4$$

(c) This number is not possible because $\sqrt{2}$ is irrational, so it containts an infinite number of non-repeating decimal digits. Because machines are finite precision, no computer can fully represent $\sqrt{2}$ without any rounding or approximation.

When converted to binary, $\sqrt{2}_2 = 1.011\ 010\ 100\ 000\ 100\ 111\ 100\ 11\cdots$. Therefore, x_l^* is the truncated form of the number, and x_r^* the truncated form of $\sqrt{2} + 2^{-23}$

$$x_l^* = 1.011\ 010\ 100\ 000\ 100\ 111\ 100\ 11,\ x_r^* = 1.011\ 010\ 100\ 000\ 100\ 111\ 101\ 00$$

$$\begin{array}{rr} & 1.01101010000010011110100 \\ - & 1.01101010000010011110011 \\ \hline & 0.00000000000000000000001 \end{array}$$

$$x_r^* - x_l^* = 2^{-23} = 1.192092895507812510^{-}7$$

7. (a) A problem is ill-conditioned if its solution is highly sensitive to small changes in the input data. True or False?

True. An ill-conditioned problem will grow a slight error in input data to a large error in later stages

(b) Using higher-precision arithmetic will make an ill-conditioned problem better conditioned. True or False?

False. While the initial input data may not be as incorrect, an ill-conditioned problem will still grow that error until the result is not significant.

(c) If two real numbers are exactly representable as floating-point numbers on a finite-precision machine, then so is their product. True or False?

False. If the two numbers are small enough, i.e. $2^{-23} * 2^{-23}$ on a single-precision machine, then their product will be smaller than the underflow limit of the machine, which cannot be exactly represented. If denormal numbers are included, this will still hold true, but the underflow limit will be much lower.

(d) Consider the sum

$$S = \frac{1}{x+1} + \frac{1}{x-1}, \quad x \neq 1.$$

For what range of values is it difficult to compute S accurately in a finite-precision system? How will you rearrange the terms in S so that the difficulty disappears?

This sum would be difficult to compute for $1 < x < 1 + \epsilon_{mach}$, and for $-1 - \epsilon_{mach} < x < -1$. When this is true, the machine cannot accurately carry out the subtraction, the significance is lost, and the result is zero. To avoid this, some algebraic manipulation is needed.

$$S = \frac{(x+1)+(x-1)}{(x+1)(x-1)} = \frac{2x}{x^2-1}, \quad x \neq 1$$

Any values of x that are extremely close to 1 will be increased to more significant numbers due to the squaring of x, thus avoiding the catastrophic collision.

(e) In a finite-precision system with UFL $= 10^{-40}$, which of the following operations will incur an underflow?

i. $\sqrt{a^2+b^2}$, with $a = 1,\ b = 10^{-25}$.

This operation will not incur an underflow. When b^2 is computed, the guard bits built into the machine will be able to represent the quantity 10^{-25}. Since the result of the operation is $1.00\cdots$, there is no underflow.

ii. $\sqrt{a^2+b^2}$, with $a = b = 10^{-25}$.
This operation will incur an underflow. Because the sum $a^2 + b^2$ must be stored before the square root is carried out, the sum must be greater than the UFL. However, $(10^{-25})^2 + (10^{-25})^2 = 2 * 10^{-50}$, so the UFL is breached.

iii. $(a \times b)/(c \times d)$, with $a = 10^{-20},\ b = 10^{-25},\ c = 10^{-10},\ d = 10^{-35}$. This operation will incur an underflow. When the initial operations are done, $(a*b), (c*d)$, the results must be stored somewhere to then be used for division. Each result will be 10^{-45}, which is below the UFL.

MATLAB Code

Problem 1:

```
% Graph f(x) = e^x*cos(2x) and its 4th degree Taylor Polynomial,
% T(x) = 1 + x - (3/2)x^2 - (11/6)x^3 - (7/24)x^4

clc

x = linspace(-pi/4, pi/4);      % domain of the functions
y = exp(x).*cos(2*x);           % actual function

t = 1 + x - (3/2)*x.^2 - (11/6)*x.^3 - (7/24)*x.^4;
                                % taylor polynomial

plot(x,y,x,t);                  % create graph

% add graph information
title('Graph of T_4(x) and f(x)');
xlabel('-\pi/4 < x < \pi/4');
legend('f(x)', 'T_4(x)');
```

Figure 4: Code for Question 1, part d

```
% Graph absolute error of Taylor expansion

clc

x = linspace(-pi/4, pi/4);                  % domain of the function

f = exp(x).*cos(2*x);                       % initial function f(x)
t = 1+x-(3/2)*x.^2-(11/6)*x.^3-(7/24)*x.^4; % Taylor polynomial T_4(x)

err = abs(f - t);                               % error function

plot(x, err);                           % plot the graph

% add graph information
title('Graph of absolute error of T_4(x)');
xlabel('-\pi/4 < x < \pi/4'); ylabel('|f(x)-T_4(x)|');

% plot max error
indexmax = find(max(err) == err);
xmax = x(indexmax);
ymax = err(indexmax);
```

Figure 5: Code for Question 1, part d

Problem 5:

```
% Graph (1-cos(x))/x^2

clc

x = linspace(-5,5);      % create domain
y = (1-cos(x))./(x.^2); % f(x)

plot(x,y);               % plot f(x) from -5 to 5
```

Figure 6: Code for Question 5